University of Illinois, Urbana Champaign
CS 598DK Special Topics in Cryptography

Instructor: Dakshita Khurana
Scribe: Surya Bakshi
Date: September 6, 2019

LECTURE

4

Pseudo-Random Generators (PRGs) and Pseudo-Random Functions (PRFs)

Last lecture we covered hard core bits and commitments. The hard core bits concept will emerge again, here, where we discuss how to construct functions that can generate pseudo-randomness, i.e. can take a finite amount of input randoness and extend to arbitrarylength strings that are computationally indistinguishable from true randomness.

From the construction that we propose for PRGs, we can construct pseudo-random functions (PRFs). At a high level, PRFs will output a string computationally indistinguishable from true randomness when given an input string. Unlike PRGs, PRFs are keyed, can be queried multiple times and must be a "function", i.e. for every input x, the output of the PRF (with a fixed key) on input x should be unique.

4.1 Preliminaries

There are a few concepts that should be restated before presenting the construction for PRGs. The first is computational indistinguishability. Computational indistinguishability means that given two distributions and a string from one of them, no polynomial time machine can distinguish which distribution the value was chosen from. More formally, recall that computational indistinguishability has the following definition:

Definition 4.1. (Computational Indistinguishability) Given two distributions X_n and Y_n, they are computationally indistinguishable if

$$\forall PPT\ \mathcal{A}, \left| \underset{x \xleftarrow{\$} X_n}{Pr}\ [\mathcal{A}(1^k, x) = 1] - \underset{x \xleftarrow{\$} Y_n}{Pr}\ [\mathcal{A}(1^k, y)] = 1 \right| \leq negl(k)$$

where k is the security parameter.

Put another way, the a probabilitic polynomial time adversary has negligible probability in deciding between the distribution X_n and Y_n. We say that $X_n \approx_c Y_n$.

For pseudo-randomness we want to create algorithms whose output is indistinguishable from the uniform distribution, U_n.

DEFINITION 4.2. (Uniform Distribution)

$$\forall \alpha \in \{0,1\}^n, Pr_{x \in U_n}[x = \alpha] = \frac{1}{2^n}$$

Note: For the remainder of the lecture notes, indistinguishablility is understood to mean computational indistinguishability.

4.2 Defining a Pseudo-Random Generator (PRGs)

As mentioned above, a pseudo-random generator expands a small bit of randomness into a large amount of pseudo-randomness. This means that the PRG is of the form $\mathsf{PRG}(k) \to k'$ where k is called the seed, $|k'| > |k|$, and $\mathsf{PRG}(k)$ is computationally indistinguishable from from a long string chosen uniformly at random. From the definition computational indistinguishability given above, the distribution of the output of a PRG should be indistinguishable from the unifrm distribution, U_n.

More formally, a PRG, G is a deterministic polynomial-time algorithm

$$G = \{G_k\}_{k \in \mathbb{N}}, G_k : \{0,1\}^k \to \{0,1\}^{m(k)} \tag{4.1}$$

such that

- $m(k) > k$, and we require that $m(k) = \mathsf{poly}(k)$
- $G_k(s)_{s \xleftarrow{\$} \{0,1\}^k} \approx_c \{u\}_{u \xleftarrow{\$} \{0,1\}^{m(k)}}$

The distribution on the right side, is another way of writing the uniform distribution, $U_{m(k)}$.

4.3 Constructing a PRG

We first start by creating a simpler PRG $G_k : \{0,1\}^k \to \{0,1\}^{k+1}$ that extends a string of length k to a pseudo-rantom string of length $k+1$.

First we must introduce the idea of length preserving one-way permutation. A one-way permutation is a function $f : \{0,1\}^k \to \{0,1\}^k$ such that $|x| = |f(x)|$ for any $x \in \{0,1\}^k$. More importantly, a one-way permutation is a bijective one-way function, i.e. it is one-to-one. The reason why we require a one-way permutation will be clear when we present the construction and the proof.

We also define the hardcore bit of the one-way function f, $B : \{0,1\}* \to \{0,1\}$. In the previous lecture we saw how to construction the function B for a one-way function B. See the preliminaries above for a definition of B.

Now we define a function

$$G_k(x)_{x \xleftarrow{\$} \{0,1\}^k} = f(x)||B(x) \tag{4.2}$$

Now we will prove that the function G_k satisfies the properties of a PRG.

Intuition. The one-way function f guarantees that if the input is drawn from the uniform distribution, the output is also the uniform distribution. We want to show that the output of G_k is also indistinguishable from the uniform distribution. The proof follows by contradiction, assuming that computational indistinguishability *does not hold* for security parameter $k+1$ and show that it would imply that an adversary can violate the definition of a hard-core bit.

Suppose the following (from the book):

$$\left|Pr_{x\in G_{k+1}}[\mathcal{A}(x)=1] - Pr_{x\in U_{k+1}}[\mathcal{A}(x)=1]\right| > negl(k+1) \tag{4.3}$$

This means that there is exists some adversary that outputs 1 with greater probability if the input is drawn from the distribution G_{k+1} than from U_{k+1}. With such distinguishability, we can see that for some bit b, there is a greater probability that $\mathcal{A}(f(x)||b) = 1$ when $b = B(x)$. This can be used to construct a predictor of the hard core bit of $f(x)$.

Formal. For some bit b, the probability the $\mathcal{A}(f(x)||b) =$ can be expanded using Bayes forumla, for $x \in U_k$ and $b \in U_1$:

$$\begin{aligned} Pr[\mathcal{A}(f(x)||b)=1] &= Pr[\mathcal{A}(f(x)||b)=1|b=B(x)]\cdot Pr[b=B(x)] \\ &\quad + Pr[\mathcal{A}(f(x)||b)=1|b=\neg B(x)]\cdot Pr[b=\neg B(x)] \\ &= \frac{1}{2}(Pr[\mathcal{A}(f(x)||b)=1|b=B(x)] + Pr[\mathcal{A}(f(x)||b)=1|b=\neg B(x)] \end{aligned}$$

Let $x := Pr[\mathcal{A}(f(x)||b)=1|b=B(x)]$ and $y := Pr[\mathcal{A}(f(x)||b)=1|b=\neg B(x)]$.
From the distinguishability assumption made, we have that

$$|Pr[\mathcal{A}(f(x)||B(x))=1] - Pr[\mathcal{A}(f(x)||b)]| = x + \frac{1}{2}(x+y)$$

$$\Rightarrow x + \frac{1}{2}(x+y) = \frac{1}{2}(x-y) > negl(k) \text{ (by the assumption)}$$

We can now show that we can create an algorithm A that can compute $B(x)$ given $f(x)$ with probability greater than $\frac{1}{2} + negl(k)$:

1. choose a random bit $b \in \{0,1\}$
2. if $\mathcal{A}(f(x)||b) = 1$ output b, otherwise $\neg b$

The algorithm uses the fact that our assumption increases the probability of $b = B(x)$ given that $\mathcal{A}(f(x)||b) = 1$. We can analyze the probablity that the algorithm successfully predicts the hard core predicate b.

$$\begin{aligned} Pr[A(f(x))=b] &= Pr[\mathcal{A}(f(x)||b)=1|b=B(x)]\cdot Pr[b=B(x) \\ &\quad + Pr[\mathcal{A}(f(x)||b)=0|b=\neg B(x)]\cdot Pr[b=\neg B(x)] \\ &= \frac{1}{2}x + \frac{1}{2}(1-y) = \frac{1}{2} + \frac{1}{2}(x-y) > \frac{1}{2} + negl(k) \end{aligned}$$

This contradicts the fact that $B(x)$ is a hard-core predicate of $f(x)$ as we are able to construct an algorithm that predicts the hard core predicate with probability greater than

a coin toss plus some neglible probability. Therefore, this shows that, in fact, the output $G_k(s) = f_k(s)||B(s) \approx_c U_{k+1}$.

With the proof above we can extend this single bit extension, we can construct an algorithm for length extension to some arbitrary polynomial $p(n)$. The algorithm is to simply run the single bit extension $p(n)$ times and use the output in a clever way. In fact, we can define the new generator

$$G_{p(n)}(s) = b_1||b_2||...||b_{p_n}$$

such that $x_0 := s$ and $x_{i+1}||b_{i+1} := G(x_i)$. What this is saying is that the last bit, the hard core bit of each extension along the way from a 1-bit extension to a $p(n)$-bit extension comprises a pseudo-random string.

The proof of indisinguishability of the arbitrarily long sequence follows directly from the definition of the hard-core bit $B(x)$ and the argument above that $f_k(x)||B(X)$ is indistinguishable from uniform randomness. The full proof is left to the textbook and the reader's consideration.

4.4 Defining Pseudo-Random Function

Before we discuss pseudo-random functions, let's define some useful notation and functions

Function Family. A function family is a map $F : \mathsf{Keys}(F) \times \mathsf{Dom}(F) \to \mathsf{Range}(F)$. In plain english, a function family is a set of functions parameterized by a key $k \in \mathsf{Keys}(F)$. The key can be though of as "selecting" a function from within the family. When we talk about ranomly choosing a key k for a function family F we mean that we are randomly choosing a function from the family F. As expcted, $\mathsf{Dom}(F)$ is the domain of F and $\mathsf{Range}(F)$ is the range of the function (review function definitions if the terms "domain" and "range" are unclea).

Generally, $\mathsf{Keys}(F) = \{0,1\}^k$, $\mathsf{Dom}(F) = \{0,1\}^l$, $\mathsf{Range}(F) = \{0,1\}^m$ for some integers $k, l, m \geq 1$. For the purposes of this lecture we will assume that a the distribution of $\mathsf{Keys}(F)$ is just the uniform distribution, and we will select random strings of size k to use as keys.

Random Function. In order to compare functions to truly random functions, we define the function family $\mathsf{Rand}^{D\to R}$to be all functions that map strings in D to strings in R. We define the set of keys for the family $\mathsf{Rand}^{D\to R}$to be all strings of length k where $D = \{0,1\}^k$. Similarly, the we define the function family $\mathsf{Perm}^D$to be the set of all permutations on the set $D = \{0,1\}^k$ for some $k \geq 1$.

Pseudo-Random Functions. A pseudo-random function is a family of functions (parameterized by some key k) whose output is computationally indistinguishable from the uniform distribution. This is different from PRGs in that the functiom must maintain and one-to-one correspondence between the inputs it gets and the outputs it provides for a given session (more on what this means later). The model we envision for PRFs is a black box model where some adversary begins a "session" with the black box function f that it can make queries to and get an answer back. After some polynomial number of queries (remember we only consider PPT adversaries) the adversary must attempt to differentiate between a truly random function (one sample from $\mathsf{Rand}^{D\to R}$, above) and a function from some

Experiment $\mathbf{Exp}^{\text{prf-1}}_{F,A}$	Experiment $\mathbf{Exp}^{\text{prf-0}}_{F,A}$
$K \xleftarrow{R} \mathsf{Keys}(F)$	$g \xleftarrow{R} \mathsf{Rand}^{D \to R}$
$d \leftarrow A^{F_K}$	$d \leftarrow A^{g}$
Return d	Return d

FIGURE 4.1: Two experiments, one with the function family F and the other with a random function g. (Bellare-Goldwasser p.61)

family F. If the outputs of the two functions are computationally indistinguishable from each other, we can say that the function family F is a pseudo-random function.

More formally, consider two worlds in which the adversary might find himself: $\mathsf{world}-0$ and $\mathsf{world}-1$

1. $\mathsf{World}-0$: the black-box function being queries is actually just a random function g sample from the set of all functions from some $D \to R$. Formally, $g \xleftarrow{\$} \mathsf{Rand}^{D \to R}$.
2. $\mathsf{World}-1$: the black-box function is actually sample from function family, F. A key is uniformaly sampled $k \xleftarrow{\$} \mathsf{Keys}(F)$ and the function being queried is then $g := F_k$.

The adversary, in this case, called the distinguisher, is trying to employ some strategy to distinguish between these two cases. The probability that the distnguisher has of "breaking" the family F is the probability that A has of distinguishing between the two worlds. Like the definition of indistinguishability, "breaking" the family F actuall means being able to distinguish between it and a random function with greater than negligible probability.
The probability of breaking is known as the adversary's *prf-advatange*.

Consider the two experiments shown in Figure 4.1. The *prf-advantage* of A is defined as

$$\mathbf{Adv}^{prf}_{F,A} = \mathbf{Pr}\left[\mathbf{Exp}^{prf-1}_{F,A} = 1\right] - \mathbf{Pr}\left[\mathbf{Exp}^{prf-2}_{F,A} = 1\right] \tag{4.4}$$

Note that this is almost identical to the definition of indistimguishability for PRGs, except in this case we are considering an experiment with queries to g and an output "guess bit".

When considering indistinguishability, we generally care about ensuring negligible probability for all PPT adversaries. Therefore, we define the *prf-advantage* of a family F as the greatest advantage than any adversary can have. Concretely, for every t, q, μ the *prf-advantage* of F is

$$\mathbf{Adv}^{prf}_{F}(t, q, \mu) = \max_{A}\left\{\mathbf{Adv}^{prf}_{F,A}\right\} \tag{4.5}$$

where the max is over all A with time-complexity t and maximum number of queries q. The sum of the lengths of the queries must be μ bits.

4.5 Constructing a PRF

There many ways to construct PRFs, however, for this lecture we will focus on constructions that use PRGs we discussed above. Specifically, the first construction that we show uses

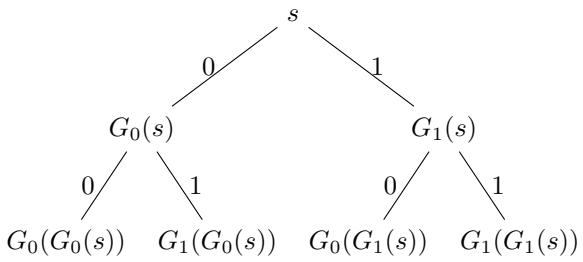

FIGURE 4.2: Illustration of the construction of the binary tree using the PRG G.

a length-doubling PRG G, where $2 \cdot |x| = |G(x)|$. This was first proposed by Goldreich, Goldwasser and Micali [1].

Recall that $G : \{0,1\}^k \to \{0,1\}^{2k}$ and that the output distribution is computationally indistinguishable from uniform randomness. Let $G_0(x)_{x \xleftarrow{\$} U_k}$ be the first k bits ouput by G, and $G_1(x)$ be bits $k+1$ to $2k$. For some arbitrary bit string $s \xleftarrow{\$} \{0,1\}^k$, $G_s(x) = G_{s_k}(\cdot\cdot(G_{s_2}(G_{s_1}(x)))\cdot\cdot)$ We can then define a function $f_k : \{0,1\}^k \to \{0,1\}^k$ in the following way:

$$f_x(s) = G_s(x) \tag{4.6}$$

For an intuitive notion, refer to Figure 4.2. The root of the tree is the original seed s. The output of the PRG on s is split into two k-bit strings. One is the left child, $G_0(s)$, and one is the right child, $G_1(s)$. Subsequently, for some intermediate node whose k-bit string is x, it's left child will be $G_0(x)$ and it's right, $G_1(x)$. The edges of the tree are labeled with the bit they correspond to: 0 is the left edge and 1 is the right edge.

On some string $t \in \{0,1\}^k$, we can traverse down the tree taking the edges corresponding to the bits in t until we reach a leaf node. There are 2^k leafs in this tree corresponding to each of the possible 2^k strings that can be passed in as input to this function. One question that might arise is: the tree is exponential in size as a function of the parameter k, so doesn't that violate the time/space constraings of the PRF? In fact, attempting to store the whole key in memory would not be possible. However, note that the algorithm for generating the tree is deterministic and the depth of the tree is k. Therefore, on some query x, we can generate the branch required *on-demand* and only take time $\log 2^k \cdot T = k \cdot T$ where T is the time it takes to run the PRG G. The running time for responding to a query is therefore polynomial in the parameter k.

We have show that such a construction can be computed efficiently. Now, we argue that this family of functions $F = \{F_k\}$ where $F_k = \{f_x\}_{x \in \{0,1\}^k}$ is a PRF.

THEOREM 4.3. *$F = \{F_k\}$ is indistinguishable from uniform randomness for all PPT adversaries A.*

Proof. Assume that F is not computationally indistinguishable from a random function. This implies there is some adversary A that can distinguish between F and a random function with non-negligible probability. We will use this advesary to construct another adversary A' that can be used to distinguish the output of the PRG G from uniform randomness.

Consider an experiment where the adversary A is querying a set of algorithms $A_i, i \in [0, k]$. Algorithm A_i creates a random binary tree of of depth k and stores random k-bit strings at level i. After level i, A_i then computes the children using the PRG G in the same way mentioned above (left child uses G_0 and right child G_1). With this construction, A_i answers the query in the same way above, it traverses the tree to reach the leaf node and returns it.

More formally, when A submits a query $x = x_1x_2...x_k$, each algorithm A_i does the following:

1. If x is the first query with some prefix $x_1x_2...x_i$, then algorithm A_i selects a random string $r \in \{0,1\}^k$, stores $(x_1, ..., x_i, r)$ and returns $G_{x_{i+1}...x_k}(r)$.
2. If this prefix has aready been seen, simply retrieve the stored pair $(x_1...x_k, v)$ and return $G_{x_{i+1}...x_k}(v)$.

What the above algorithm is doing is generating the remainder of the tree from input index $i+1$ to k using some randomly selected value r. Each A_i does the same things, but beginning at different levels i.

Now, assume a polynomial $p(x)$ that bounds the number of queries that A can make on some input k, and some set S_k of $p(x)$ $2k$-bit strings. Adversary A' functions it 2 steps. The first step it samples a number $i \xleftarrow{\$} [0...k-1]$. In the second step A' runs the algorithm A with input k. A' simulates the adversary A and interacts with a running session of A, playing the part of the oracle, and answers a query $y_1y_2...y_k$ from A as follows:

1. If y is the first query with the prefix $y_1y_2...y_i$, then pick a string $s = s_0||s_1 \xleftarrow{\$} S_k$ where $|s_0| = |s_1|$. Store the pairs $(y_1y_2...y_i1, s_1)$ and $(y_1y_2...y_i0, s_0)$ for the left and right children. Depending on the value of y_{i+1}, return $G_{y+i+1...y_l}(s_0)$ if $y_{i+1} = 0$ and $G_{y_{i+2}...y_k}(s_1)$ otherwise.
2. if the prefix has been seen before, simply get the stored pair $(y_1y_2...y_{i+1}, v)$ as above and return $G_{y_{i+2}...y_k}(v)$.

Similar to the description of A_i, this algorithm is also generating the tree based on a prefix of the input and a value selected out of the set S_k. The key to this proof is showing that what S_k actually is. If S_k is just randomly selected strings of size $2k$ then A' is essentially simulating A_{i+1} (for the same i). If you recall, at level i, A_i chooses a random value as the string and generates the remainder of the tree with it. If S_k is generated by G then we have that A' is a simulation of A with algorithm A_i instead.

Given our assumption that such a distinguisher, A, exists, the adversary A' constructed here is able to distinguish betwee the cases where S_k is uniformly random and generated by G, contraditing the fact that the output of G should be indistinguishable from the uniform distribution.

□

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov. The proof for the PRF construction provided relies heavily on the a paper by Goldreich, Goldwasser and Micali [1]. The proof for the construction was not

provided in class, therefore this paper provided the outline for the adversary and the proof as presented here.

References

[1] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. 1986.